Weaving Business Processes and Rules: A Petri Net Approach

Jian Yu[1], Quan Z. Sheng[1], Paolo Falcarin[2], and Maurizio Morisio[2]

[1] School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia
jian.yu01@adelaide.edu.au, qsheng@cs.adelaide.edu.au
[2] Dipartimento Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino
{paolo.falcarin,maurizio.morisio}@polito.it

Abstract. The emerging service-oriented computing paradigm advocates building distributed information systems by chaining reusable services instead of by programming from scratch. To do so, not only business processes, but also business rules, policies and constraints need to be encoded in a process language such as Web Services Business Process Execution Language (WS-BPEL). Unfortunately, the intermixing of business processes and rules in a single process weakens the modularity and adaptability of the systems. In this paper, we propose a formal approach to model the weaving of business processes and rules, following the aspect-oriented principle. In particular, we use Predicate/Transition (PrT) nets to model business processes and business rules, and then weave them into a coherent PrT net. The resulting woven nets are ready for analysing system properties and simulating system behaviour.

Keywords: Business process modelling, business rules, aspect-orientation, Petri nets.

1 Introduction

Service-oriented computing (SOC) builds on the software engineering trends of greater encapsulation and composing rather than programming [1]. It's why business processes play a central role in SOC: using distributed, platform-independent, and well-encapsulated services as basic building blocks, business processes organize and coordinate the behaviour of services to achieve business goals in a loosely coupled and flexible manner. In the case of Web services, Web Services Business Process Execution Language (WS-BPEL, BPEL for short) has been considered as a de facto industry standard to create composite service applications.

However, the flow logic encoded in processes cannot represent the complete features of a business: there are also rules, policies and constraints manifesting the decision aspect of the business. In fact, business rules are used extensively in some decision-intensive domains such as finance and insurance sectors to model

J. Yang et al. (Eds.): UNISCON 2009, LNBIP 20, pp. 121–126, 2009.

and document business requirements. A serious problem appears if we implement business rules using a process-oriented paradigm where business rules are mixed and coded in the process logic as a whole monolithic block. As a result, the original modularity of business rules is lost and it becomes hard for business rules to change without affecting the core composition logic [2].

In this paper, we use Predicate/Transition nets (PrT nets) [3], a kind of widely used high-level Petri nets, to model both business processes and business rules, and then use an aspect-oriented mechanism to weave them into a coherent PrT net. Our approach not only keeps the modularity of business rules, but also supports formal verification thanks to the well-established theoretical foundation of Petri nets.

The rest of this paper is organized as follows: Section 2 briefly overviews some fundamental concepts and definitions that underpin the discussion throughout the paper. Section 3 explains how to model business rules with PrT nets. Section 4 explains the PrT net-based aspect and the weaving mechanism, and Section 5 concludes the paper.

2 Business Rules and Aspect-Orientation

In this section, we briefly review the concepts of business rule and aspect-orientation. Some example rules and aspects used throughout the paper are also discussed.

According to the Business Rules Group [4], a business rule is a statement that defines or constrains some aspect of a business. It is intended to assert business structure or to control the behaviour of the business. In [5], business rules are classified into four types: *constraint rule*, *action-enabler rule*, *computation rule*, and *inference rule*.

For instance, a travel-package-requesting scenario could have the following business rules [2]:

R_1(*constraint rule*): a vacation request must have a departure airport and a destination airport.
R_2(*action-enabler rule*): if no flight is found, do not look for accommodation.
R_3(*computation rule*): if more than 2 persons travel together, give 10% discount to the total price.
R_4(*inference rule*): if a customer is frequent customer, he gets a discount of 5%.

The reason why R_4 is an inference rule is that to resolve what is a frequent customer, we need another two rules:

R_5(*constraint rule*): if a customer has bought more than 5 travel packages, he is a frequent customer.
R_6(*constraint rule*): if a customer has bought products for a sum exceeding 4000 euros, he is a frequent customer.

Aspect-orientation is a strand of software development paradigm that models scattered crosscutting system concerns as first-class elements, and then weaves them together into a coherent model or program.

Referring to Aspect4J [6], an aspect-oriented program usually consists of base modules and a number of aspects that modularizes crosscutting concerns. An *aspect* wraps up *pointcuts* and *advices*. A pointcut picks out certain *join points*, which are well-defined points (e.g., method calls) in the program flow. An advice is a piece of code that is executed when a join point is reached. There are three types of advices:

- A *before advice* runs just before the join points picked out by the pointcut.
- An *after advice* runs just after each join point picked out by the pointcut.
- An *around advice* runs instead of the picked join point.

Following gives an example to illustrate the idea of aspect-orientation. Supposing we have a simple business process containing three sequentially executing services, say *getFlight*, *getAccommodation*, and *calculatePrice*, to apply all the business rules ranging from R_1 to R_4 on this process, we first define the three services as pointcuts so that whenever a service is called, a join point is reached where advices can take effect. Then we define four advices: R_1 *before getFlight*, R_2 *around getAccommodation*, R_3 *after calculatePrice*, and R_4 *after calculatePrice*. The above pointcuts and advices can be grouped as an aspect. This aspect means: R_1 should be satisfied before executing *getFlight*; and if the condition of R_2 is true, *getAccommodation* will not be executed; and R_3 and R_4 should be executed after *calculatePrice*.

3 Modeling Business Rules with PrT Nets

In this section, we show our idea of how to use PrT nets to model the above-mentioned four types of business rules. Specifically, we use the terms *constraint nets*, *action-enabler nets*, *computation nets* and *inference nets* to call the nets representing corresponding business rules, and call them *rule nets* in general.

Syntactically, we classify the transitions in a rule net with three stereotypes:

- An *action transition* T_A represents a business activity, e.g., get flight.
- A *computation transition* T_C represents the specific action of assigning variables with arithmetic expressions, e.g., assigning $price \times 90\%$ to the variable *price*.
- And a *dummy transition* T_D does nothing.

Transition stereotypes provide additional information to a net at the business level; they do not change the behavioural semantics of rule nets.

To model a constraint rule, we use a dummy transition with its inscription representing the constraints. For example, R_1 can be modelled as the constraint net in Fig. 1.

To model an action-enabler rule, we use an action transition to represent the business activity, and the inscription of this transition representing the enabling condition.

To model a computation rule, we use a computation transition with its name representing the computation expression. Note that the computation expression

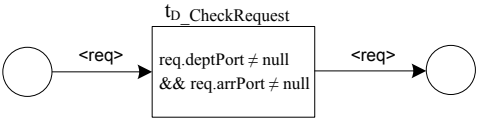

Fig. 1. Constraint net R_1

is only reflected in the name of the transition since just like the execution of a business action, the interpretation of expressions is outside the scope of PrT nets.

An inference rule is created by composing two rule nets with AND-JOIN, OR-JOIN, or SEQ-JOIN operators. Intuitively, if we connect two rule nets with AND-JOIN, then the composed rule net means the conjunction of the two rules; if we connect two rule nets with OR-JOIN, then the composed rule net means the disjunction of the two rules. Note that AND-JOIN and OR-JOIN are also two workflow patterns defined in [7]. The sequential join of of two rule nets means that the resolve of one rule net depends on the consequents of the other rule net. The $SEQ - JOIN$ operation fuses the source place of the dependent net with the sink place of the independent rule net. Usually, an inference net can be built first by introducing the net representing the final goal, and then introducing the rules backwards with the SEQ-JOIN operation based on the cause-effect relations until all the newly introduced rules are resolvable.

Taking R_4 as an example, to resolve the meaning of frequent, we need another two rules: R_5 and R_6. Fig. 2.a, b and c are the rule nets for R_4, R_5 and R_6 respectively. Because R_4 depends on the consequent of R_5 or R_6, we use OR-JOIN to compose them and then use SEQ-JOIN to connect the combined consequent to the rule net of R_4 to form a complete inference rule net as depicted in Fig. 2.d.

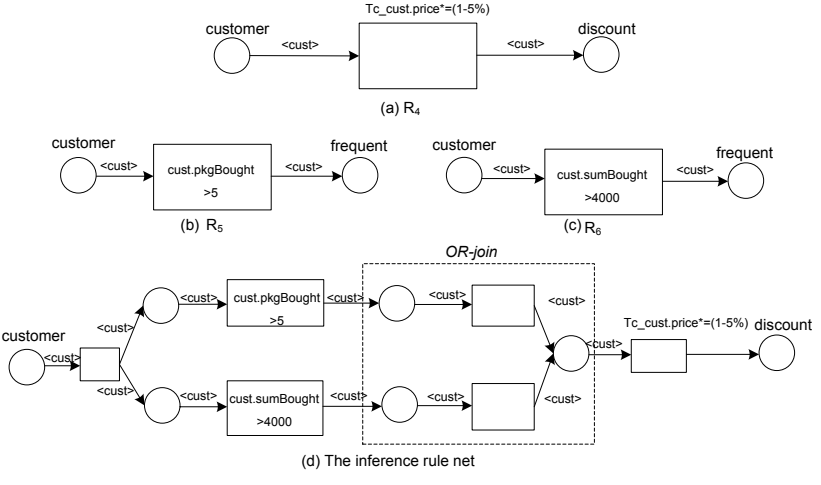

Fig. 2. The inference rule derived from R_4, R_5, and R_6

4 Weaving Rule Nets into Process Nets

Corresponding to rule nets, we use the term *process nets* to call the PrT nets representing business processes. For example, Fig. 3 is the process net N for the travel package request process described in Section 2.

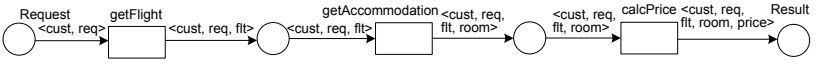

Fig. 3. Travel package request process net N

Just like aspect-oriented programming, we use *pointcut* to select a transition as join point. Without losing generality, every pointcut can only select one transition to simplify the definition of weaving. The advices are represented by rule nets. A rule net can be weaved into a process net either $before$, $after$, or $around$ a pointcut/transition. Another important concept is context net, for exposing context of the process net to the rule net. It is interesting to note that the process net and its rule nets could be authored independently, which reflects the modular feature of aspect-orientation. To weave them smoothly, we use context net

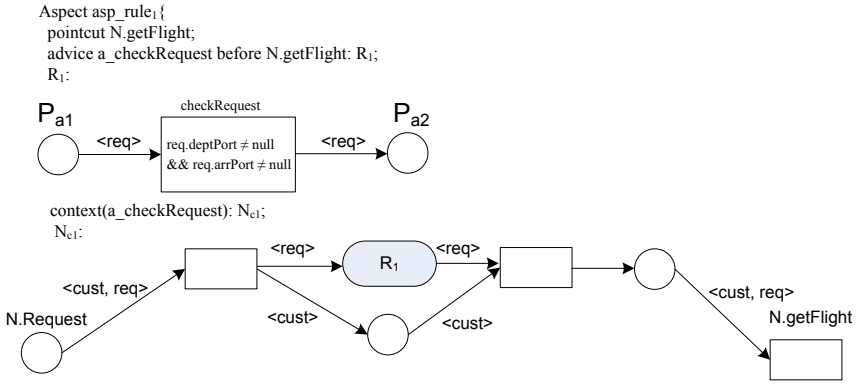

Fig. 4. Aspect asp_rule_1

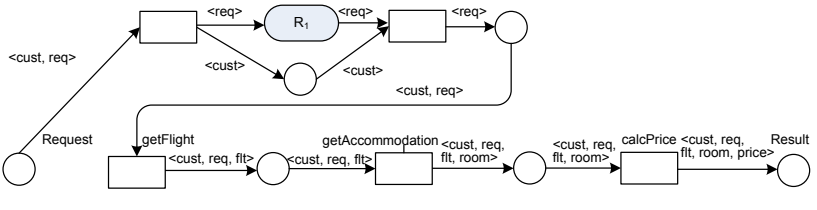

Fig. 5. Resulting net of weaving N and asp_rule_1

to put them in the same context and mediate possible parameter inconsistencies. For example, the input parameters for the transition $N.getFlight$ is a structure $< Customer, Request >$, but its constraint net R_1 accepts $< Request >$ as input. Finally, we can wrap up pointcuts, advices, and context net into an *aspect*.

To demonstrate our idea, we give an example on how to weave the constraint net R_1 before $N.getFlight$. Fig. 4 presents an aspect asp_rule_1. The context net is built first by introducing the pre-set of the pointcut, i.e. transition $N.getFlight$, as the initial context, and then splitting the request to match the input of the advice net, finally merging the results and transferring them back to the cutting point. Note that we use an round rectangle to represent the net R_1 for simplicity.

Fig. 5 is the resulting net after we weave asp_rule_1 into N. This net is built simply by putting the context net and the process net together, and then deleting the arcs between the cutting point and its preset because it is a $before$ cut. Note that elements with the same names are merged to reflect the expansion of context.

5 Conclusion

In this paper, we present a PrT net-based approach to weaving business processes and rules. The woven net not only keeps the modularity of rules, but also is ready for analysis and simulation by various PrT techniques and tools. We view our work presented in this paper as a first step towards a formalized model-driven approach of developing service-oriented information systems.

References

1. Yu, Q., Bouguettaya, A., Medjahed, B.: Deploying and Managing Web Services: Issues, Solutions, and Directions. The VLDB Journal 17(3), 537–572 (2008)
2. Charfi, A., Mezini, M.: Hybrid Web Service Composition: Business Processes Meet Business Rules. In: 1st International Conference on Service Oriented Computing ICSOC, pp. 30–38 (2004)
3. Genrich, H.J.: Predicate/Transition Nets. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 207–247. Springer, Heidelberg (1987)
4. The Business Rules Group: Defining Business Rules, What Are They Really?, `http://www.businessrulesgroup.org`
5. von Halle, B.: Business Rules Applied: Building Better Systems Using the Business Rules Approach. Wiley, Chichester (2001)
6. Gradecki, J.D., Lesiecki, N.: Mastering AspectJ: Aspect-Oriented Programming in Java. Wiley, Chichester (2003)
7. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)